OpenMP Offloading for the NVPTX target in GCC

Alexander Monakov

amonakov@ispras.ru

Institute for System Programming of Russian Academy of Sciences

GNU Tools Cauldron 2016

PTX

PTX: abstract ISA for NVIDIA GPUs
Abstracted away:

- Registers
- Calling convention
- Stack pointer

Execution model

- *Thread/Lane:* one execution context.
  Private storage: registers, stack (`.local`) memory.
- *Warp:* 32 lanes executing in lockstep. Fast cross-lane exchange insn.
- *CTA/Block:* a group of warps. H/w barrier insn.
  Private storage: `.shared` memory.
- *Grid:* a group of blocks.

Warp synchronicity

SIMT: single instruction – multiple threads

- Warp members share the program counter
- Taking divergent branches handled by executing both targets in sequence, with different *lane mask*
- Reconvergence instructions are implicitly inserted at postdominators

Implicit masking constrains possible control flow

- Mutex with lane granularity implemented as a function call will deadlock

Mapping to OpenMP

- Teams $\leftrightarrow$ blocks.
- Threads $\leftrightarrow$ warps.
- SIMD lanes $\leftrightarrow$ PTX lanes.

Outside of OpenMP-SIMD regions, warps execute in a way that all lanes are active, but observable behavior is as if any lane was the only active one:

- Side effects (atomics, syscalls) happen once per warp
- Auto storage is per-warp
- Register store same values

Changes for OpenMP/PTX

- NVPTX backend additions

  `7 files changed, 594 insertions(+), 107 deletions(-)`

- libgomp port

  `58 files changed, 2610 insertions(+), 624 deletions(-)`

- middle-end changes (mostly omp-low.c)

  `1 file changed, 515 insertions(+), 89 deletions(-)`

Most middle-end changes are for OpenMP SIMD

The challenge

OpenMP lowering is producing IR that is compiled:

- For the host CPU via normal flow
- For the accel arch via offload LTO streaming

SIMD needs different lowering for host-vs-accel

- Host: OpenMP-SIMD relies on autovectorization and internal `GOMP_SIMD_nn()` functions
- PTX: OpenMP-SIMD implemented via SIMT

The idea is to arrange that SIMD-via-SIMT is lowered in a way that is easily collapsed back after IPA passes

Example – basic SIMD loop

```
#pragma omp simd
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;
```

ompdevlow for host:

- GOMP_SIMT_LANE() $\rightarrow 0$
- GOMP_SIMT_VF() $\rightarrow 1$

Example – basic SIMD loop

```
#pragma omp simd
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;
```

ompdevlow for host:

- GOMP_SIMT_LANE() → 0
- GOMP_SIMT_VF() → 1

Example – basic SIMD loop

```
#pragma omp simd
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;
```

ompdevlow for host:

- GOMP_SIMT_LANE() $\rightarrow$ 0
- GOMP_SIMT_VF() $\rightarrow$ 1

Example – safelen

```
#pragma omp simd safelen(SAFE)
for (i = START; i < END; i += STEP)
  BODY;
```

```
if (SIMT_LANE() < SAFE)
  for (i = START + STEP * SIMT_LANE();
        i < END;
        i += STEP * MIN(SAFE, SIMT_VF()))
    BODY;
```

Example – lastprivate

```
#pragma omp simd lastprivate(v)
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;

i -= STEP * (SIMT_VF() - 1);
cond = !(i < END);
if (GOMP_SIMT_VOTE_ANY(cond))
  {
    lane = GOMP_SIMT_LAST_LANE(cond);
    v = GOMP_SIMT_XCHG_IDX(v, lane);
  }
```

Example – reduction

```
#pragma omp simd reduction(+:v)
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;

for (t = 1; t < SIMT_VF(); t <<= 1)
  v += GOMP_SIMT_XCHG_BFLY(v, t);
```

This requires that reduction op is commutative

Sync NaN payloads

```
if (v != v)
  v = GOMP_SIMT_XCHG_IDX(v, 0);
```

Example – reduction

```
#pragma omp simd reduction(+:v)
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;

for (t = 1; t < SIMT_VF(); t <<= 1)
  v += GOMP_SIMT_XCHG_BFLY(v, t);
```

This requires that reduction op is commutative

Sync NaN payloads

```
if (v != v)
  v = GOMP_SIMT_XCHG_IDX(v, 0);
```

Example – reduction

```
#pragma omp simd reduction(+:v)
for (i = START; i < END; i += STEP)
  BODY;
```

```
for (i = START + STEP * SIMT_LANE();
     i < END;
     i += STEP * SIMT_VF())
  BODY;

for (t = 1; t < SIMT_VF(); t <<= 1)
  v += GOMP_SIMT_XCHG_BFLY(v, t);
```

This requires that reduction op is commutative

Sync NaN payloads

```
if (v != v)
  v = GOMP_SIMT_XCHG_IDX(v, 0);
```

Example – ordered SIMD

```
#pragma omp simd
...
#pragma omp ordered
  STMT;
```

```
for (t = SIMT_LANE();
     GOMP_SIMT_VOTE_ANY(t >= 0);
     t--)
  if (GOMP_SIMT_ORDERED_PRED(t))
    STMT;
```

Outlining SIMD regions

Need to outline SIMD regions into separate functions

- Switching from per-warp stacks to per-lane stacks
- No way to model stack switch in IR

Current plan:

- Reuse task/parallel region outlining in omp-low.c
- Mark as inlinable ASAP after LTO stream-in on host

```
#pragma omp simd
  ...
```

becomes in gimplification

```
#pragma omp parallel _simtreg_
#pragma omp simd
  ...
```

Uninitialized auto variables

UB in C11

```
#include <stdio.h>

int main()
{
    /* register */ int n;

    printf("%d\n", n);
}
```

Reading an automatic variable that could have been declared with the `register` keyword invokes undefined behavior.

No UB

```
#include <stdio.h>

int main()
{
    int n;
    &n;
    printf("%d\n", n);
}
```

Reading an automatic variable that (formally) has its address taken results in an indeterminate value *without invoking UB*.

Uninit vars

Branching on an uninit var outside of SIMD region:

```
#pragma omp target
{
  int n;
  if (n)
    puts("Fizz");
  else
    puts("Buzz");
}
```

could execute both branches.

Uniform SIMT

```
    atom.op dest, ...
```

becomes

```
@p  atom.op  dest, ...
    shfl.idx dest, dest, m
```

where

- `m = (laneid & unisimt_mask)`
- `p = (laneid == m)`